Stable

Python Development

Common Patterns and Anti-Patterns

Larger Software Packages (Like DiFX)

- Anaconda:
  - Python 2 and 3 versions.
  - Freely redistributable.
  - Very easy installation.
  - Includes most common packages: NumPy, SciPy, Pandas, etc.
  - Embed and freeze just the packages and versions you use (smaller install).
  - Maintenance handled by Anaconda Cloud.
  - Conda package manager makes for easy extension.
- Conda:
  - Install additional modules/packages.
  - Example: `conda install -c anaconda netcdf4`

Python, 2 or 3?

Python 2

- **More common** (now).
- **SciPy modules** are almost **all available**.
- Some newer **web modules** are **missing**.
- **File IO** is **very easy**, but **missing some advanced features**.

Python 3

- **Growing popularity**.
- Some **SciPy libraries** are **missing**.
- Most newer **web modules available**.
- **File IO** is slightly **more complicated**, but has more advanced features.

Just a script?
Code for any (recent) version of Python!

- If possible, code scripts to run in **both Python 2 and 3**.
- Suggestion: Target **Python 2.7** and **Python 3.4**.
- Check documentation for “**New in version** X.X.X.”
- Check versions with `sys.version_info`.

Python 2/3 Gotchas

- **`print`** should always use parentheses with a single argument.

```
print('string of text')
print(5)
print('')
```

- Use `basestring` for python2 and `str` for python3.

```
import sys
_py3 = sys.version_info >= (3,)
_str = str if _py3 else basestring
```

- Avoid opening in binary mode!

```
f = open('foo.bin', 'rb') # don't do this!
```

  - Python 3 will read in `bytes`, but Python 2 will read in `str`.
  - Read binary files with **NumPy** instead, and use text mode for text files.
- Watch for Unicode issues!
  - Python 2 uses `unicode` for Unicode strings (`u'foo'`), and `str` for others (`'foo'`).
  - Python 3 uses `str` (which is now Unicode) for all strings.
  - If you *need* 8-bit string(s) to work, use explicit newline and error handler:

```
open('foo.bin', newline='\n', errors='surrogateescape')
```

Dealing with modified Julian datetimes (MJD)

```
import datetime

epoch_mjd = datetime.datetime(1858, 11, 17)

def datetime2mjd(datetime):
    '''
    Convert (datetime.datetime) datetime to (float) modified Julian datetime.

    return    (float) modified Julian datetime
    datetime  (datetime.datetime) datetime to convert
    '''
    return (datetime - epoch_mjd).total_seconds() / 86400

def mjd2datetime(mjd):
    '''
    Convert (float) modified Julian datetime to (datetime.datetime) datetime.

    return  (datetime.datetime) datetime to convert
    mjd     (float) modified Julian datetime
    '''
    return epoch_mjd + datetime.timedelta(mjd)
```

Performance Testing

- Linux tools work for scripts:

```
time -v python my_script.py
```

- Quick-and-dirty method is `time`:

```
import time
t0 = time.time()
do_stuff()
print('that took ' + str(time.time() - t0) + ' sec')
```

- Easiest method is the `timeit` module:

```
import timeit
#              code to run                setup code          times to run
timeit.timeit('my_module.do_thing()', 'import my_module', number=10000)
```

- More complete test of script:

```
python -m cProfile my_script.py arg1 arg2 …
```

Hot Opinions:

- Tabs or Spaces?
  PEP prefers **4 spaces**, but **tabs are fine** too. **Pick one** and stick to it.
- Single, double, triple, or double-triple quotes?
  Triple for documentation:`'''Doc string here.'''`
  Double if singles are inside:`"string with 'string' inside"`
  Single for other strings: `'Normal string here.'`